Faults Discovery Using Mined Data

Charles Lee
SAIC
c/o NASA Ames Research Center
Moffett Field, CA 94035
650-604-6054
charles.lee@saic.com

Abstract—Fault discovery in the complex systems consist of model based reasoning, fault tree analysis, rule based inference methods, and other approaches. Model based reasoning builds models for the systems either by mathematic formulations or by results of experiments. Fault Tree Analysis shows the possible causes of a system malfunction by enumerating the suspect components and their respective failure modes that may have induced the problem. The rule based inference build the model based on the expert knowledge. Those models and methods have one thing in common; they have presumed some prior-conditions. Complex systems often use fault trees to analyze the faults. Fault diagnosis, when an error occurs, is performed by engineers and analysts performing extensive examination of all data gathered during the mission. International Space Station (ISS) control center operates on the data feedback from the system and decisions are made based on threshold values by using fault trees. Since those decision-making tasks are safety critical and must be done promptly, the engineers who manually analyze the data are facing the challenge of time limit. To automate this process, this paper presents an approach that uses decision trees to discover faults from data in real-time and capture the contents of fault trees as the initial state of the trees.

1. Introduction

Decision trees (also called classification trees) are the binary trees built from data samples and can classify the objects into different classes. In our case, the decision trees can classify different fault events or normal events. Given a set of data samples, decision trees can be built and trained, and then by running the new data through the trees, classification and prediction can be made. In this way, diagnostic knowledge for fault detection and isolation can be represented as diagnostic rules; we call this tree the diagnostic decision tree(DDT). By showing the fault path in decision trees, we also can point out the root cause when a fault occurs. Since all the procedures and algorithms are available to build decision trees, the trees built are cost effective and time effective. Because the diagnostic decision trees are based on available data and previous knowledge of subsystem logic, the DDT can also be trained to predict faults and detect unknown faults. Based on this, the needs for on-board real time diagnostics can readily be met. Diagnostic Decision Trees are built based on the fault trees as static trees that serve as the fundamental diagnostic trees, and the dynamic DDTs are built over time from vehicle telemetry data. The dynamic DDT will add the functionalities of prediction, and will be able to detect unknown faults. Crew or maintenance engineers can use the decision tree system without having previous knowledge or experience about the diagnosed system. To our knowledge, this is the first paper to propose a solution to build diagnostics decision trees from fault trees, which convert the reliability analysis models to diagnostic models. We show through mapping and ISS examples that the approach is feasible and effective. We also present future work and development.

Detecting faults in a complex system requires complex diagnostic tools. Fault tree is one of those tools to be used for analyzing and diagnostic fault. The fault tree concept was introduced by Bell Telephone Laboratories in 1962 for the U.S. Air Force for use with the Minuteman system [8]. It was later adopted and extensively applied by the Boeing Company and is one of the most widely used methods in system reliability analysis for a long time [4]. It is a deductive procedure for determining the various combinations of hardware and software failures and human errors that could result in the occurrence of specified undesired events (referred to as top events) at the system level. As part of the analysis, the minimal cut sets of a fault tree can be determined [3], and then fault trees can be built. Individual fault trees can be visualized and drawn. Fault trees are usually individually built for each part of the system for each top event. It is very difficult to have generic software to traverse fault trees. On the other hand, the decision trees are matured data structures and it is very easy to be manipulated in a software program. Using decision trees to represent fault trees will increase the operability and decrease response time for system diagnostics, and furthermore, will make it easier for users to visualize the root cause of the fault and path from which the fault came. The high availability of many different tree algorithm implementations in the computer science field makes using decision tree to manage the fault trees one of best approaches. In this paper, we present a method to convert existing fault trees to decision trees. More general ways of constructing decision trees are presented. The method is easy to program and run on a computer since the decision tree algorithms have many available implementations [5]. This method also provides a good tool for researchers on simulation and prediction tasks. By using this method, one can analyze data samples from the past and categorized them into different classes; abnormal, normal, and fault events in such a way that future faults can be predicted from the past. This could be a data mining components with run time updating processes. Such an artificial intelligent application is presented in the paper as a form of framework architecture.

2. DETECTING FAULTS FROM DATA

Using decision trees, we can detect fault from data by monitoring the data values and compare to the fault patterns. When we do not have the fault patterns at beginning of the system, we need to set up initial trees with results of fault analysis of static system. Usually we can start with fault trees. Then we migrate the fault trees to decision trees. We also can build decision trees from events and telemetry data during run time when real data is coming in. The decision trees converted from fault trees can be used as diagnostic tools. When fault occurs, we can recognize the fault by running telemetry data through the trees and finding out where the data stops. Also, the type of fault can be categorized and its path can be determined by such trees. Other applications are also possible by utilizing decision trees. Once we know that the decision tree is very well suited for data mining tasks, we can apply our trees to a data mining framework targeting at recognizing fault patterns and do early fault detection and prediction. Data mining is the process of analyzing data from different perspectives and summarizing it into useful information. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. In our case, it is the process of finding fault patterns. A design model, also we can call it a framework model, is presented in Figure 1. In the figure, we can see how the hybrid decision trees, in this case it a kind of fault decision tree, fit into the knowledge discovery part of the data mining process [7]. Initially, the trees are built for known fault. For example, the initial trees converted from fault trees. Then we have decision trees built from real time data while the system is running. While we know the fault trees could not be developed run time and could not be used in such application, the decision trees are so easy to be fit into such an application. We can build such a tree that records fault patterns each time a fault event occurs. Essentially, we record the fault trends patterns so we can use such trees to recognize a fault in its early stages.

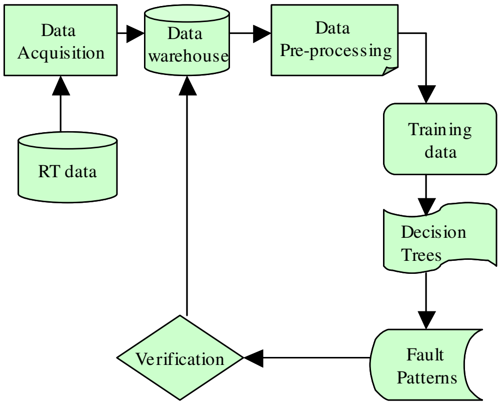

Figure 1 Decision trees in fault discovery application

3. CONVERSION METHOD

There are some other attempts to represent fault trees by other forms; one of them is building diagnostic maps from fault trees [2]. Decision trees are trees usually built from data. Let's look at a fault tree and see how can we map it to the decision tree. Take a sample fault tree in the form of following:

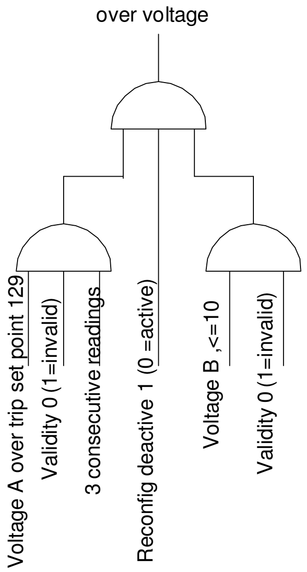

Figure 2 Over Voltage Event Fault Tree

We have 6 inputs to the tree. The inputs remain the same for the decision tree. The corresponding decision tree should have the same functionality in terms of sample inputs and fault triggers. In the other words, the same inputs to the fault tree, or to the decision tree will have the same result. The corresponding tree can be built as in Figure 3. If by applying the telemetry data one can propagate all the way to terminal node 6, we know that the system is at an over voltage fault. The decision tree not only provides the final result if the system is at fault status, it can also provides the interim status by looking at where the sample data end up. To ease the decision tree generation and notation, we give the signals short names as follows:

Consecutive readings = ct; Voltage A over trip point 129 = Ua; Validity UaA = UdA; Reconfig deactive = Config; Voltage B = Ub; Validate Ub = UdB.

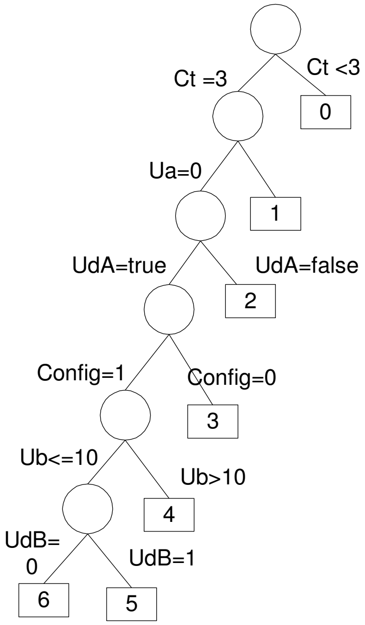

Figure 3 Decision Tree for Over Voltage

To show a more common case that includes an OR gate in the fault tree, another example is shown below:

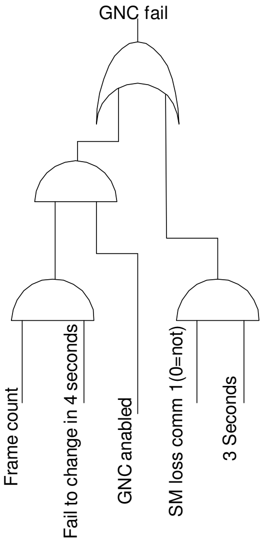

Figure 4 GNC Fail Event Fault Tree

Similarly, we represent the signals n short notations. Frame count = Fc; Fail to change in 4 seconds = Fc4; GNC enabled = GNCe; SM loss conn = Smloss; 3 second = SM3. In this case, more balanced data inputs will end up with more evenly distributed decision trees. In the same way as we showed earlier, the corresponding decision tree is presented as follows:

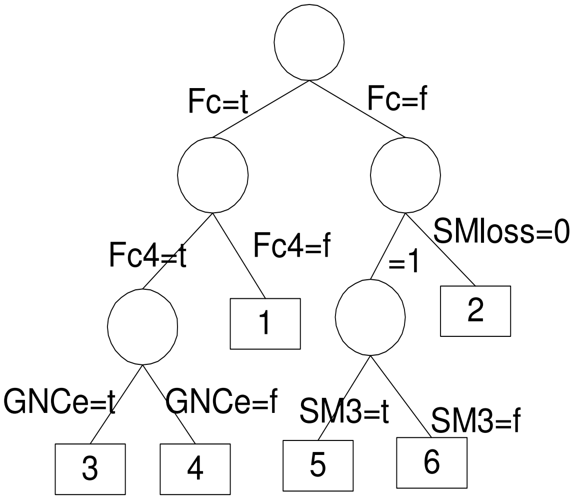

Figure 5 Decision Tree for GNC Fail

We had demonstrated that fault trees can be mapped to decision trees with examples of the over voltage and GNC fail fault trees. The convenience use of decision tree use is that the available decision tree software programs can easily pin point a root cause of an event (including fault event) by recording the edges in the path of the tree when giving reports and evaluation of the system status. For more general purposes and for ease of illustration, we can abstract the information into a map and construct a decision tree from this map. The map basically represents the different events, including fault events, in the n dimension space. When we deal with decision trees, we call the events classes. The class could be fault, nominal, or warning etc. This demonstration shows that the decision tree not only can be derived from the fault tree but also can be constructed from data samples, which is very useful in real time fault detection and prediction.

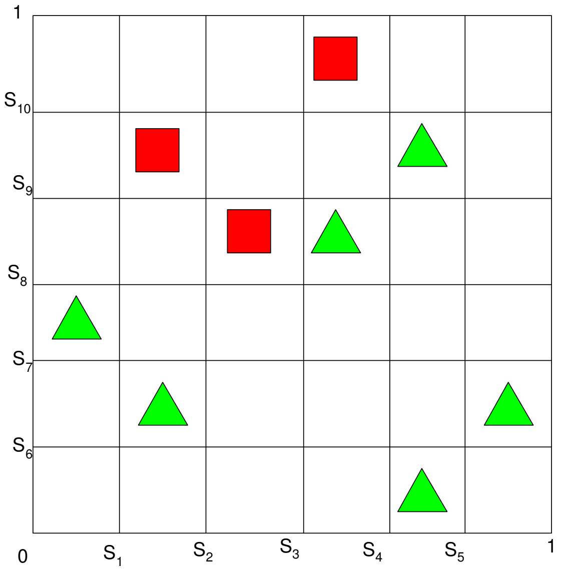

Figure 6 Class distributions

To map to the decision tree, we use an example to explain it. We assume faults in the two dimension space for simplification of visualization. Multiple dimensions will follow the same rules. In Figure 5, we give an example of the fault scenarios, the triangles represent normal, called Class 1 and squares represent faults, called Class 2. We will use cumulative distribution function (*cdf*) for tree construction. The *cdf* is the probability that the variable takes a value less than or equal to *x*. That is

$$F(x) = Pr[X \leq x] = \alpha \qquad (1)$$

This can be expressed mathematically for continuous distribution:

$$F(x) = \int_{-\infty}^{x} f(\mu)d\mu \qquad (2)$$

For a discrete distribution, the *cdf* can be expressed as

$$F(x) = \sum_{i=0}^{x} f(i) \qquad (3)$$

When we construct a decision tree, we have a root, then we have two branches, further, each of those branches can, have a maximum of two branches, until no further branches can be constructed and we reach the bottom of the tree and we are done. Those procedures could be said in another way; we are splitting the data until it couldn't be split any more. So what we need is information on where to split, and when do we stop splitting. With the method of accumulated distribution function, we construct the trees using the following steps. First, calculate the *cdf* for each class. Second, compare the result for each class at all the points. Third, the largest value will be picked and the maximum value of x will be the split point. Repeat first to third step until no more points to split remain. In the example, we calculate the *cdf* for each class as a function of each attribute (see Figure 5), and then pick the split point where the difference of the two *cdf* values is maximum.

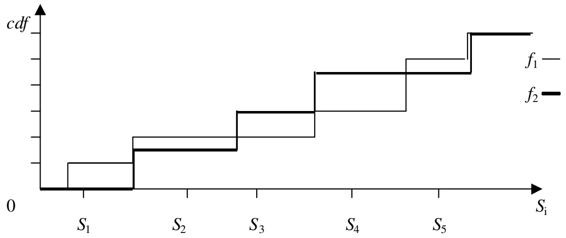

Figure 7 Calculate split point by using *cdf*.(1)

We repeatedly split until all samples in a node are of the same class. In Figure 6, the horizontal axis is the possible split points $S_i \ 1 < i <= 5$ corresponding to the *x*-axis in Figure 6, and the vertical axis is the value of the *cdf* for each class. In Figure 6 f_1 is the *cdf* value for class 1 (cylinders) and f_2 is the *cdf* value for class 2 (cubes).

In Figue 6, the vertical axis is the possible split points. $S_i \ 5 < i <= 10$ corresponding to the y-axis in Figure 6, and the horizontal axis is the value of the *cdf* for each class. In Figure 6 f_1 is the *cdf* value for class 1 (cylinders) and f_2 is the *cdf* value for class 2 (cubes). The purpose is to find the point where the distance between f_1 and f_2 is the maximum. To calculate the *cdf*, we used the estimated function n_i / N_i. The total number of samples in class 1 is 6, and in class 2 is 4. N_1 = 6 and N_2 = 4. As shown in Figure 5 at split point S_1 , we have the f_1 value of 1/6, and f_2 value of 0. At split point S_2 we have the f_1 value of 2/6, and a f_2 value of 1/4. At split point S_3 we have the f_1 value no change, still 2/6 or 1/3, and a f_2 value of 2/4 or 1/2. At split point S_4 we have the f_1 value of 1/2 and a f_2 value of 3/4. At split point S_5 we have the f_1 value of 5/6 and a f_2 value of no change, 3/4.

Similarly in Figure 8 at split S_6 the value of f_1 is 1/6 and f_2 is 0. At split point S_7 we have the f_1 value of 3/6, and a f_2 value of 0. With the same calculations, at split point S_8 we have the f_1 value of 4/6 and a f_2 value of 0. Again, at split point S_9 we have the f_1 value of 5/6 and a f_2 value of 1/4. At split point S_{10}, we have the f_1 value of 1 and a f_2 value of 2/4.

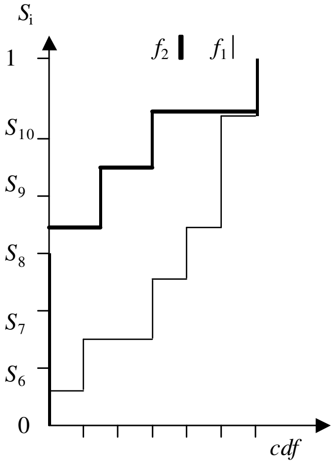

Figure 8 Calculate Split Point Using *cdf* (2)

From Figures 7 and 8, we can see that the split S_8 has the maximum distance (4/6) between f_1 and f_2 among all others. Therefore, we pick the first split point as S_8. After we split the set on S_8, we have two subsets, one of the subsets has only class 1 in it, and so we don't need to do the further split on this subset. But on the other subset, we will repeat the same calculations on the remaining samples to find the further split points. The procedures to calculate the cdf and select the maximum distance between f_1 and f_2 are the same as above. The constructed tree is shown in Figure 9.

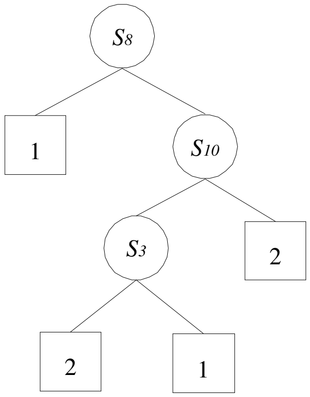

Figure 9 Final Decision Tree

When the real time data comes in, we let them propagate through the decision tree that we constructed. The faults are detected and classified when the data samples fall into a fault class at the terminal node. To illustrate how the fault happened, we can show the fault mode by tracking the path that the data went through. Visualization of the path with a distinctive color or shape will show the user the clear cause of the fault.

This method not only can apply to the conversion of the fault trees to decision trees, it can also construct decision trees from data samples at run time during operation of ISS over time. By simply selecting a set of data samples from time to time, we can build decision trees dynamically. In later time, the built decision trees can be used to compare the new data to the old data and to predict future faults. The best use of such trees is to build trees by applying grouped fault scenarios and then applying real time data to the tree to compare the pattern to know fault patterns, faults can be detected when a pattern is matched.

4. CONCLUSION

We started from ISS fault tree examples to migrate to decision trees by presenting a method for converting fault trees to decision trees. The method shows that the visualization of root cause of faults is easier and that tree manipulation becomes more programmatic via available decision tree programs. The visualization of decision trees for diagnostics shows a format that is straightforward and easy to understand. For ISS real time fault diagnostics, the status of the systems could be shown by running the signals through the trees and watching where it stops. The other advantage to using decision trees is that the trees can learn the fault patterns and predict future faults from the historic data. The learning is done not only on the static data sets but also can be done at runtime; through accumulating the real time data sets, the decision trees can gain and store faults patterns in the trees and recognize them when they reoccur. The decision tree plays the role in knowledge discovery while the fault tree could not.

5. FUTURE DEVELOPMENT

This paper presented the method to migrate the fault trees to decision trees, which lays a good foundation for using a data mining technique in advanced diagnostic systems. The next step will naturally fall to a project to implement data mining software for fault detection, prediction, and analysis. Such software will use the decision trees as an engine inside of the diagnostic system application. This engine will be able to gain knowledge of fault patterns then recognize faults when they reoccur.

REFERENCE:

[1] Ramesh K. Rayudu, Sandhya Samarasinghe, and Don Kulasiri, "A Comparison of Model-based Reasoning and Learning Approaches to Power Transmission Fault Diagnosis," 2nd New Zealand Two-Stream International Conference on Artificial Neural Networks and Expert Systems (ANNES '95), pp 218, 1995.

[2] Tariq Assaf and Joanne Bechta Dugan, "Automatic generation of diagnostic expert systems from fault trees," Reliability and Maintainability Symposium, January 2003.

[3] Zhihua Tang and Joanne Bechta Dugan, "Minimal Cut Set and Sequence Generation for Dynamic Fault Trees," Reliability and Maintainability Symposium, January 2004.

[4] Joanne Bechta Dugan, "Software system analysis using fault trees," Chapter 15, *Handbook of Software Reliability Engineering*, editor MR. Lyu, IEEE Computer Society Press, McGraw-Hill Publication 1996.

[5] J. R. Quinlan, Induction of Decision Trees, Machine Learning, v.1 n.1, p.81-106, 1986

[6] Ping. Li, Richard. E. Haskell, Darrin. M. Hanna, "Optimizing Fuzzy Decision Tree by Using Genetic Algorithms," Proceedings of the International Conference on Artificial Intelligence, June, 2003, Las Vegas, USA

[7] Olaru, C. and L. Wehenkel, *Data Mining*. IEEE Computer Applications in Power, 1999. **12**(3): p. 19-25.

[8] N. H. Robert and D. F. Haasl, *Fault Tree Handbook*, National Technical Information Service, Springfield, VA, 1981

[9] Leiguang Gong, Doug Riecken, "Constraining Model-based reasoning using context,", IEEE/WIC International Conference on Web Intelligence (WI'03), pp 507, October 13 - 17, 2003 Halifax, Canada